Cancelling a Windows Runtime asynchronous operation, part 6: C++/WinRT-generated asynchronous operations

devblogs.microsoft.com/oldnewthing/20200708-00

July 8, 2020

Raymond Chen

Last time, we learned that C++/WinRT defers to the underlying asynchronous operation to report the cancellation in whatever way it sees fit. Today, we'll look at the case that the asynchronous operation was generated by the C++/WinRT library.

When you invoke the `Cancel()` on a C++/WinRT asynchronous operation, this is the code that runs:

```
struct promise_base : ...
{
    ...

    void Cancel() noexcept
    {
        winrt::delegate<> cancel;
        {
            slim_lock_guard const guard(m_lock);
            if (m_status == AsyncStatus::Started)
            {
                m_status = AsyncStatus::Canceled;
                cancel = std::move(m_cancel);
            }
        }
        if (cancel)
        {
            cancel();
        }
    }
};
```

The promise transitions into the `Canceled`, and if the coroutine had registered a cancellation callback, it is invoked.

Whenever the coroutine associated with the promise performs a `co_await`, the `await_transform` kicks in (sorry, I haven't explained this yet, but trust me), and that's where the C++/WinRT library gets a chance to abandon the operation:

```
template <typename Expression>
Expression&& await_transform(Expression&& expression)
{
    if (Status() == AsyncStatus::Canceled)
    {
        throw winrt::hresult_canceled();
    }
    return std::forward<Expression>(expression);
}
```

The thrown `hresult_canceled` exception is captured into the operation for later rethrowing.

The C++/WinRT library also checks for cancellation when the coroutine runs to completion:

```
struct promise_type final : ...
{
    ...

    void return_void()
    {
        ...
        if (this->m_status == AsyncStatus::Started)
        {
            this->m_status = AsyncStatus::Completed;
        }
        else
        {
            WINRT_ASSERT(this->m_status == AsyncStatus::Canceled);
            this->m_exception = make_exception_ptr(winrt::hresult_canceled());
        }
        ...
    }
};
```

If the operation has been cancelled, then we manufacture a fake `hresult_canceled` exception and save it in the `m_exception`.

So we see that whether the operation's cancellation is detected by `await_transform` or by `return_void` (or `return_value` for coroutines that produce a value), we end up with an `hresult_canceled` exception stashed in the operation.

And it is this exception that comes back out when somebody asks for the result of the asynchronous activity:

```
struct promise_type final : ...
{
    ...

    void GetResults()
    {
        ...
        if (this->m_status == AsyncStatus::Completed)
        {
            return;
        }
        this->rethrow_if_failed();
        ...
    }

    void rethrow_if_failed() const
    {
        if (m_status == AsyncStatus::Error || m_status == AsyncStatus::Canceled)
        {
            std::rethrow_exception(m_exception);
        }
    }
};
```

If the operation was canceled, then we reach `rethrow_if_failed` which rethrows the captured exception, which we saw earlier is going to be an `hresult_canceled`.

But C++/WinRT is not the only source of `IAsyncAction` and `IAsyncOperation` objects. Next time, we'll look at another major source: WRL.

Raymond Chen

Follow